ABF++: Fast and Robust Angle Based Flattening

Alla Sheffer, Bruno Lévy, Maxim Mogilnitsky, Alexander Bogomyakov

ABF++: Fast and Robust Angle Based Flattening
Temporary version - In print

Alla Sheffer
University of British Columbia
Bruno Lévy
Inria Lorraine
Maxim Mogilnitsky and Alexander Bogomyakov
Technion

Conformal parameterization of mesh models has numerous applications in geometry processing. Conformality is desirable for remeshing, surface reconstruction, and many other mesh processing applications. Subject to the conformality requirement, these applications typically benefit from parameterizations with smaller stretch. The Angle Based Flattening (ABF) method, presented a few years ago, generates provably valid conformal parameterizations with low stretch. However, it is quite time consuming and becomes error prone for large meshes due to numerical error accumulation. This work presents ABF++, a highly efficient extension of the ABF method that overcomes these drawbacks, while maintaining all the advantages of ABF. ABF++ robustly parameterizes meshes of hundreds of thousands and millions of triangles within minutes. It is based on two main components: (1) a new numerical solution technique that dramatically reduces the dimension of the linear systems solved at each iteration, speeding up the solution; (2) an efficient hierarchical solution technique. The speedup with (1) does not come at the expense of greater distortion. The hierarchical technique (2) enables parameterization of models with millions of faces in seconds, at the expense of a minor increase in parametric distortion. The parameterization computed by ABF++ are provably valid, i.e. they contain no flipped triangles. As a result of these extensions, the ABF++ method is extremely suitable for robustly and efficiently parameterizing models for geometry processing applications.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Color, shading, shadowing, and texture*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling; G.1.6 [**Numerical Analysis**]: Optimization—*Constrained optimization*; J.6 [**Computer Aided Engineering**]:

General Terms: Algorithms

Additional Key Words and Phrases: mesh processing, parameterization, conformality.

Author's address: A. Sheffer, Department of Computer Science, University of British Columbia, Vancouver, BC , V6T 1Z4, Canada e-mail:sheffa@cs.ubc.ca
Author's address: B. Levy, INRIA Lorraine, 545000 Vandoeuvre, France, e-mail: levy@loria.fr
Author's address: M. Mogilnitsky, Department of Computer Science, Technion, Haifa, 32000, Israel
Author's address: A. Bogomyakov, Department of Computer Science, Technion, Haifa, 32000, Israel, e-mail:alexb@cs.technion.ac.il

1. INTRODUCTION

With recent advances in computer graphics hardware and digital geometry processing, parameterized surface meshes have become a widely used geometry representation. The parameterization defines a correspondence between the surface mesh in 3D and a 2D domain, referred to as the *parameter space*. In the general case, the paramerizations are expected to be bijective, i.e. one-to-one. However for most practical applications a weaker requirement of local bijectivity, is sufficient. Local bijectivity is achieved when the planar mesh has no flipped (inverted) triangles. In the context of this paper the term validity implies local bijectivity. The principal uses of parameterization are texture mapping and geometry editing:

- ◇ *Texture mapping* is the oldest application of parameterization. The parameter space is covered with an image, which is then mapped onto the model through the parameterization. With the introduction of programmable GPUs, more general attributes can be mapped onto the model in real time (e.g., BRDFs, bump maps, displacement maps,...). It is even possible to completely represent the geometry of the model in parameter space, leading to the *geometry images* approach [Gu et al. 2002].
- ◇ *Geometry Editing* is the second, increasingly popular application domain. Using parameterization, it is possible to replace complex 3D algorithms operating on the surface with much simpler 2D computations performed in parameter space. Applications that benefit from parameterized representation include multiresolution editing [Lee et al. 1998], surface fitting [Hormann and Greiner 2000], mesh morphing [Praun et al. 2001], remeshing [Alliez et al. 2003] and extrapolation [Levy 2003], to name just a few.

For all of these applications, the quality of the result depends heavily on the amount of deformation caused by the parameterization. In the ideal case, areas and angles are preserved through the mapping, i.e. the parameterization is *isometric*. To reach this goal, the approach described in [Maillot et al. 1993] minimizes a matrix norm of the deformation tensor. Unfortunately, only a small class of surfaces, i.e. *developable* surfaces, can be isometrically parameterized. Therefore, depending on the application, existing parameterization methods attempt to minimize different distortion components, such as angle deformation (conformal/harmonic parameterizations), length deformation (stretch), or area deformation.

Previous work

Floater and Hormann [Floater and Hormann 2004] provide an extensive survey of the state of the art in parameterization research. Below we briefly review the major techniques proposed for planar parameterization. We refer the reader to [Floater and Hormann 2004] for a more detailed discussion of the numerous techniques available.

For many geometry processing applications, such as remeshing and surface reconstruction, the preservation of shape (angles) during mapping is of major concern. Angle preservation is typically addressed either from the harmonic point of view (Dirichlet energy) or from the conformal point of view (Cauchy-Riemann equation). In the context of computer graphics, the first discrete version of harmonic maps was proposed in [Eck et al. 1995]. [Desbrun et al. 2002] used a discretization of the Dirichlet energy suggested in [Pinkall and Polthier 1993] to construct free-boundary harmonic maps. [Gu and Yau 2002] used the same discretization formula to approximate the Laplace Beltrami operator. The main drawback of all of these methods is that triangle flips can happen in the presence of obtuse

angles, breaking the local bijectivity requirement on the mapping. The harmonic mapping method described in [Floater 1997] is based on Tutte's barycentric mapping theorem [Tutte 1960] and does not suffer from this limitation. A bijective mapping is guaranteed, provided that the mesh boundary is fixed to a convex polygon. A simpler approximation of harmonicity is proposed in [Floater 2003]. The alternative, conformal perspective is used by [Lévy et al. 2002]. The authors use a discretization of the Cauchy-Riemann equation for constructing free-boundary maps. The discrete formulation of conformal energy they propose is equivalent to [Desbrun et al. 2002] and hence suffers from the same triangle flip problem [Lévy et al. 2003]. Note that in general harmonic and conformal maps are not identical [Floater and Hormann 2004].

Fixed (convex) boundary approaches such as [Eck et al. 1995; Floater 1997; 2003] typically generate significantly more distortion than free-boundary techniques. However, only a few free-boundary conformal parameterization methods are guaranteed to avoid triangle flips. The MIPS method [Hormann and Greiner 2000; Hormann 2001] minimizes a non-linear function of the first fundamental form of the mapping. The method is time consuming and the results demonstrated in the paper are limited to parameterizations of simple surfaces with near-convex boundaries. [Degener et al. 2003] use another function of the first fundamental form to measure conformality. Using a state of the art iterative hierarchical solver, they report times of 5 minutes for parameterizing meshes with 60K faces.

The Angle Based Flattening (ABF) method [Sheffer and de Sturler 2001] uses a very different approach from most other techniques. It defines an angle preservation metric directly in terms of angles. It first computes the parameterization in angle space and only then converts it into 2D coordinates. In addition to avoiding flips, its important advantage is that in addition to closely preserving the angles it typically produces parameterizations with low area (and stretch) deformation (see Figure 1). This is particularly noticeable when comparing the results of ABF to linear, free-boundary techniques [Desbrun et al. 2002; Lévy et al. 2002]. In Section 5.2 we discuss the causes for this different behaviour. In addition to the advantages mentioned above, [Sheffer and de Sturler 2001] describe a simple post-processing procedure which can be used to eliminate overlaps in the parameterization. However, since the optimization procedure used by ABF is numerically expensive, and due to numerical errors occurring when reconstructing the 2D coordinates from the angles, ABF becomes impractical for meshes with more than 30K faces. [Liesen et al. 2001] discuss methods to speed-up ABF but do not provide an implementation of these. [Zayer et al. 2004] recently proposed a different strategy for solving the non-linear optimization problem defined by ABF. Their method requires a couple of minutes to parameterize medium sized models (10K faces). We will study ABF in depth in Section 2, and propose new techniques to overcome the approach's limitations.

Several authors proposed parameterization techniques for area/stretch preservation during mapping. In [Desbrun et al. 2002] a local measure of area preservation was introduced. Aiming at optimally mapping a signal onto the surface, [Sander et al. 2001] and [Sander et al. 2002] minimize a non-linear stretch metric. The method is particularly well suited for texture mapping. Similarly to [Hormann and Greiner 2000; Degener et al. 2003] the authors use a hierarchical solver to speed up the non-linear optimization.

Several recent papers address the trade-off between angle and stretch/area deformations [Desbrun et al. 2002; Degener et al. 2003; Yoshizawa et al. 2004]. This is typically

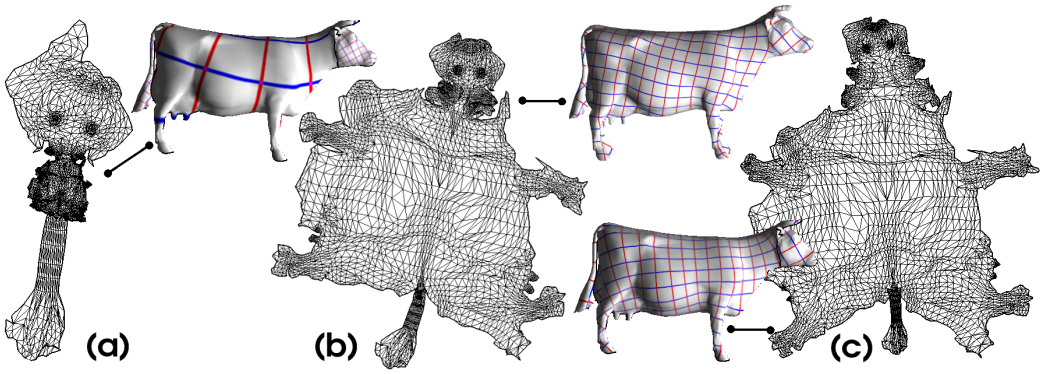


Fig. 1: *Parameterization comparison: (a) [Levy et al. 2002]/[Desbrun et al. 2002] - the linear formulation provides an angle preserving parameterization, but introduces significant stretch (Error metrics* $E(\alpha) = 0.00075$, $L_2^{stretch} = 99.3$, *and* $L_2^{shear} = 0.013$ *- the metrics are explained in Section 6); (b) Stretch minimization [Sander et al. 2001] (* $E(\alpha) = 0.0017$, $L_2^{stretch} = 1.032$, *and* $L_2^{shear} = 0.156$*); (c) ABF (* $E(\alpha) = 0.0006$, $L_2^{stretch} = 1.096$, *and* $L_2^{shear} = 0.072$*). The ABF result combines good angle preservation with low stretch. The runtimes are given in Table I (Cow has 6K faces).*

achieved by introducing energy functionals such as those described above for each deformation component and minimizing their combined functional (sum or product).

To speedup the parameterization process for large models, many authors propose hierarchical parameterization techniques [Sander et al. 2001; Ray and Lévy 2003; Hormann 2001; Degener et al. 2003; Aksoylu et al.] which use mesh multiresolution structures. Our work combines sophisticated numerical tools with a multiresolution approach to achieve maximal speedup and generate angle preserving low stretch parameterizations of huge meshes.

1.1 Overview

This paper introduces ABF++ - an extension of the angle preserving ABF method for parameterizing large meshes. It consists of two complimentary techniques, *direct ABF++* suitable for parameterizing medium to large meshes and *hierarchical ABF++*, for parameterizing huge meshes with millions of triangles.

The direct ABF++ method generates provably valid (no flipped triangles), conformal, low stretch parameterizations of meshes with several hundred thousand faces in a couple of minutes by using two new tools:

- ◇ A new solution mechanism based on algebraic transforms reduces the dimension of the Hessian used in ABF by a factor of five. This reduction results in an improvement of up to $10\times$ in speed. Since speedup is achieved through solely numerical manipulations, it does not come at the cost of increased parametric distortion.

The second component of ABF++, the hierarchical ABF++ parameterization scheme, is used to further speedup the parameterization procedure and parameterize models of millions of triangles. The scheme uses direct ABF++ to parameterize a simplified mesh. Then it proceeds to compute the parameterization for the full model, using a local relaxation scheme utilizing a multiresolution hierarchy. The scheme is carefully tailored to parameterize huge meshes in second.

Both techniques have significant advantages compared to existing methods. The direct

ABF++ is significantly more efficient and robust than previous non-linear conformal parameterization techniques. While the direct ABF++ is slower than linear free-boundary conformal methods, it introduces significantly less stretch. The hierarchical ABF++ is an order of magnitude faster than the direct version. As demonstrated by the examples (Section 5), it is 4 to 5 times faster than the fastest free-boundary technique with which we are familiar [Ray and Lévy 2003]. At the same time it computes valid parameterizations with only slightly higher parametric distortion than direct ABF++.

The rest of the paper is organized as follows. Section 2 reviews the standard ABF technique. Section 3 describes the novel solution mechanism that uses sequential linearly constrained programming and algebraic transformations to speedup the parameterization computation. Section 4 describes our hierarchical solution technique. Section 5 demonstrates the results of direct and hierarchical ABF++ parameterization. It provides a comparison of the two methods to other popular techniques in terms of both distortion and speed (Section 5.1) and discusses (Section 5.2) the reasons for the differences in the distortion. Finally, Section 6 summarizes the presented research.

2. THE ABF METHOD - BRIEF REVIEW

The Angle Based Flattening (ABF) method [Sheffer and de Sturler 2001] is based on the observation that the set of angles of a 2D triangulation uniquely defines the triangulation up to global scaling and rigid transformations. Building on this observation, ABF first computes the parameterization in angle space and then converts it to 2D coordinates. The angle space formulation makes this technique particularly suitable for reducing the angular distortion of the mapping.

2.1 Formulation

In angle space the minimized function is simply

$$E(\alpha) = \sum_{t \in T} \sum_{k=1}^{3} \frac{1}{w_k^t} (\alpha_k^t - \beta_k^t)^2. \tag{1}$$

where α_k^t are the unknown planar angles and β_k^t are the optimal angles. The index t goes over the set T of triangles in the mesh and the index k goes over the angles in each triangle. The weights w_k^t are set to $\frac{1}{\beta_k^{t\,2}}$ to reflect *relative* rather than *absolute* angular distortion. To prevent degenerate configurations of the angles they are sometimes scaled during the solution (details can be found in [Sheffer and de Sturler 2001]).

To provide a set of values that defines a planar parameterization, a number of constraints are incorporated into the solution:

◇ Triangle validity (for each triangle):

$$\forall t \in T, \quad C_{Tri}(t) = \alpha_1^t + \alpha_2^t + \alpha_3^t - \pi = 0; \tag{2}$$

◇ Planarity (for each interior vertex):

$$\forall v \in V_{int}, \quad C_{Plan}(v) = \sum_{(t,k) \in v^*} \alpha_k^t - 2\pi = 0, \tag{3}$$

where V_{int} is the set of interior vertices and v^* is the set of angles incident on vertex v.

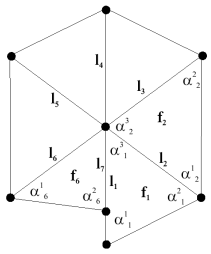


Fig. 2: *A (sub)mesh generated without enforcing the reconstruction constraint.*

◇ Reconstruction (for each interior vertex) - this constraint ensures that edges shared by pairs of triangles have the same length (Figure 2):

$$\forall v \in V_{int}, \quad C_{Len}(v) = \prod_{(t,k)\in v^*} \sin\alpha^t_{k\oplus 1} - \prod_{(t,k)\in v^*} \sin\alpha^t_{k\ominus 1} = 0. \tag{4}$$

The indices $k\oplus 1$ and $k\ominus 1$, respectively, indicate the next and previous angles in the triangle.

2.2 ABF Solution Mechanism

The resulting constrained minimization problem is formulated using Lagrange multipliers $(\lambda_{Tri}, \lambda_{Plan}, \lambda_{Len})$. The augmented objective function F is:

$$F(x) = F(\alpha, \lambda_{Tri}, \lambda_{Plan}, \lambda_{Len}) = E + \sum_t \lambda^t_{Tri} C_{Tri}(t) + \sum_v \lambda^v_{Plan} C_{Plan}(v) + \sum_v \lambda^v_{Len} C_{Len}(v).$$

Sheffer and de Sturler [Sheffer and de Sturler 2001] minimize the (non-linear) augmented objective function F using Newton's method, as follows:

$$\begin{array}{l} \textbf{while} \quad \|\nabla F(x)\| > \varepsilon \\ \quad \textbf{solve} \quad \nabla^2 F(x)\delta = -\nabla F(x) \\ \quad x \leftarrow x + \delta \\ \textbf{end} \end{array} \tag{5}$$

The size of the Hessian matrix $\nabla^2 F(x)$ is $4n_f + 2n_{int}$ where $n_f = |T|$ is the number of mesh triangles and $n_{int} = |V_{int}|$ is the number of interior vertices. There are $3n_f$ variables, and $n_f + 2n_{int}$ Lagrange multipliers. The linear system $\nabla^2 F(x)\delta = -\nabla F(x)$ is solved using a sparse direct linear solver (SuperLU [Demmel et al. 1999]).

[Zayer et al. 2004] proposed simplifying the solution process by applying the log function to the reconstruction constraint (Equation 4), replacing the product by a sum. This results in a much simpler matrix structure of $\nabla^2 F(x)$. The downside of this conversion is that the matrix becomes ill-conditioned and hence the system cannot be stably solved by direct solvers. Using an iterative solver instead, the authors quote times of 237 seconds for a model of 25K faces. Given such times, the iterative procedure is actually slower than our implementation of the original solution technique using SuperLU, which takes half of this time to parameterize models twice the size (Table I).

3. SPEEDING-UP ABF

This section introduces a solution technique that dramatically improves the performance of the angle based parameterization. To speedup the solution process we will first simplify the system solved by each Newton iteration, and then find a much smaller system to solve.

3.1 Sequential Linearly Constrained Programming

The ABF formulation is based on constrained minimization of a quadratic form. The quadratic form is very simple, since its optimum is already known (the optimal angles β), while the constraints (Equation 4 in particular) are rather complex. To overcome this complexity we propose to use sequential linearly constrained programming [Nocedal and Wright 2000]. This technique for solving constrained minimization problems considers the constraints as linear at each iteration. In other words, it neglects the terms coming from the second order derivatives of the constraints in the Hessian matrix $\nabla^2 F(x)$ (see [Nocedal and Wright 2000]). This simplifies the system solved at each iteration of the non-linear solver (5) at the expense of a slightly increased number of iterations.

The linear system $\nabla^2 F(x)\delta = -\nabla F(x)$ solved at each step thus becomes:

$$\begin{bmatrix} \Lambda & J^t \\ J & 0 \end{bmatrix} \begin{bmatrix} \delta_\alpha \\ \delta_\lambda \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \quad \text{where:} \tag{6}$$

$$\Lambda = \text{diag}\left(\frac{2}{w_k^t}\right), \quad J = \begin{bmatrix} \dfrac{\partial^2 F}{\partial \lambda_i \partial \alpha_k^t} \end{bmatrix}, \quad b_1 = -\nabla_\alpha F, \quad b_2 = -\nabla_\lambda F.$$

Since the constraints are considered linear, the upper-left bloc (Λ) is now a simple *diagonal* matrix. The lower-right block is null. Based on this particular matrix structure it is now possible to dramatically reduce the dimensions of the matrix inverted at each iteration.

3.2 First Matrix Split

System 6 can be rewritten as follows:

$$\Lambda\delta_\alpha + J^t\delta_\lambda = b_1 \tag{7}$$

$$J\delta_\alpha = b_2. \tag{8}$$

It is now feasible to separately compute the step vector δ_λ for the Lagrange multipliers, and express the step vector δ_α for the variables as a function of δ_λ:

$$J\Lambda^{-1}J^t\delta_\lambda = b^* \qquad \text{where} \quad b^* = J\Lambda^{-1}b_1 - b_2 \tag{9}$$

$$\delta_\alpha = \Lambda^{-1}(b_1 - J^t\delta_\lambda) \tag{10}$$

The first line (9) is obtained by multiplying (7) by $J\Lambda^{-1}$ and substituting $J\delta_\alpha$ using (8). The second line (10) is obtained by multiplying (7) by Λ^{-1}. Note that since Λ is diagonal computing Λ^{-1} is trivial. Using these expressions, the algorithm (5) can be rewritten as follows:

while $\|\nabla F(x)\| > \varepsilon$
  compute b, J, Λ
  solve Equation 9 $\rightarrow \delta_\lambda$
  $\delta_\alpha \leftarrow \Lambda^{-1}(b_1 - J^t\delta_\lambda)$ (Equation 10)
  $\lambda \leftarrow \lambda + \delta_\lambda$; $\alpha \leftarrow \alpha + \delta_\alpha$ $/*x = (\alpha, \lambda)*/$
end

The initial linear system of dimension $4n_f + 2n_{int}$ has been replaced by a smaller linear system of dimension $n_f + 2n_{int}$, where the matrix $J\Lambda^{-1}J$ depends only on the Jacobian of the constraints J and the diagonal Λ (see Equation 9). Solving the system gives the step vector δ_λ for the Lagrange multipliers, and it is easy to compute the step vector δ_α for the variables from δ_λ (see Equation 10). Since the size of the linear system solved at each iteration is much smaller than in the initial algorithm, this algorithm is much faster. We now show that it is possible to reduce the size of the system even further, by analyzing the structure of the matrix $J\Lambda^{-1}J^t$ and applying a similar kind of substitution.

3.3 Second Matrix Split

To analyze the particular structure of $J\Lambda^{-1}J^t$, we can split the Jacobian of the constraints J into two sub-matrices J_1 and J_2:

$$J \quad = \quad \begin{bmatrix} J_1 \\ J_2 \end{bmatrix}, \tag{11}$$

where $J_1(n_f \times 3n_f)$ is the Jacobian of C_{Tri} constraints and $J_2(2n_{int} \times 3n_f)$ is the Jacobian of the C_{Plan} and C_{Len} constraints. Note that J_1 has a very simple structure:

$$J_1 \quad = \quad \begin{bmatrix} 1\,1\,1 & & & 0\,0\,0 \\ & 1\,1\,1 & & \\ & & \ddots & \\ 0\,0\,0 & & & 1\,1\,1 \end{bmatrix}.$$

Moreover, its rows are orthogonal and linearly independent. We now decompose $J\Lambda^{-1}J^t$ as follows:

$$J\Lambda^{-1}J^t \quad = \quad \begin{bmatrix} \Lambda^* & J^{*t} \\ J^* & J^{**} \end{bmatrix} \quad \text{where} \quad \begin{cases} \Lambda^* & (n_f \times n_f) & = & J_1\Lambda^{-1}J_1^t \\ J^* & (2n_{int} \times n_f) & = & J_2\Lambda^{-1}J_1^t \\ J^{**} & (2n_{int} \times 2n_{int}) & = & J_2\Lambda^{-1}J_2^t \end{cases} \tag{12}$$

Using the bloc decomposition of the matrix $J\Lambda^{-1}J^t$, Equation 9 becomes

$$\begin{bmatrix} \Lambda^* & J^{*t} \\ J^* & J^{**} \end{bmatrix} \begin{bmatrix} \delta_{\lambda_1} \\ \delta_{\lambda_2} \end{bmatrix} \quad = \quad \begin{bmatrix} b_1^* \\ b_2^* \end{bmatrix}. \tag{13}$$

In other words,

$$\Lambda^*\delta_{\lambda_1} + J^{*t}\delta_{\lambda_2} \;=\; b_1^* \tag{14}$$
$$J^*\delta_{\lambda_1} + J^{**}\delta_{\lambda_2} \;=\; b_2^*. \tag{15}$$

Since J_1 is orthogonal and Λ^{-1} is diagonal, the matrix $\Lambda^* = J_1\Lambda^{-1}J_1^t$ is diagonal. We can now express δ_{λ_2} independently as the solution of a linear system:

$$\left(J^*\Lambda^{*-1}J^{*t} - J^{**}\right)\delta_{\lambda_2} = J^*\Lambda^{*-1}b_1^* - b_2^*. \tag{16}$$

Equation 16 was obtained by multiplying (14) by $J^*\Lambda^{*-1}$, then substituting $J^*\delta_{\lambda_1}$ using (15). Consequently, the vector δ_{λ_1} can be computed as a function of δ_{λ_2}:

$$\delta_{\lambda_1} = \Lambda^{*-1}\left(b_1^* - J^{*t}\delta_{\lambda_2}\right) \tag{17}$$

by multiplying (14) by Λ^{*-1}.

Note that computing δ_{λ_2} requires solving a linear system of dimension $2n_{int}$, whereas the dimension of the initial problem is $4n_f + 2n_{int}$. Based on the Euler formula $n_f \approx 2n_{int}$, therefore $4n_f + 2n_{int} \approx 10n_{int}$. Hence, the proposed matrix manipulations result in a factor of five reduction in the size of the linear system solved at each iteration of the non-linear solver.

3.4 ABF++ Solution Mechanism

Using these matrix splitting expressions, the solution procedure can be rewritten as follows:

while $\|\nabla F(x)\| > \varepsilon$
  compute b, J, Λ
  solve Equation 16 $\rightarrow \delta_{\lambda_2}$
  compute δ_{λ_1}(Equation 17)
  compute δ_α(Equation 10)
  $\lambda_1 \leftarrow \lambda_1 + \delta_{\lambda_1}$; $\lambda_2 \leftarrow \lambda_2 + \delta_{\lambda_2}$; $\alpha \leftarrow \alpha + \delta_\alpha$ $/* x = (\alpha, \lambda) */$
end (18)

Compared to the original Newton formulation (5), the new method requires several additional iterations to converge (typically 8 to 10 instead of 5). However, at each iteration a five times smaller matrix is inverted. We found that using the SuperLU direct solver for solving Equation 16 gives the best results in terms of performance. This is consistent with other recent research [Sorkine et al. 2003; Sumner and Popovic 2004], which consistently indicates that direct solvers out-perform the more popular iterative techniques. The algorithm is often more than ten times faster than the original. As a result, models on the order of 100K triangles can now be parameterized in a minute or two. An additional advantage is a reduction in the memory size required to store the matrices. The results section (Section 5) compares the efficiency of the parameterization with and without the proposed speedup.

4. HIERARCHICAL PARAMETERIZATION

The direct ABF++ performs well for meshes with up to 300K faces. For larger meshes solving the linear system becomes quite time consuming. More importantly, with the increase in the size of the stored matrices, memory becomes the bottleneck of the process. Hence to efficiently parameterize huge meshes with hundreds of thousands and millions of triangles we propose a hierarchical parameterization procedure.

The basic idea of a hierarchical (or multi-resolution) approach is to reduce the problem size, then solve the smaller problem, and finally derive the solution to the original problem, using the multi-resolution hierarchy. In this work we follow this approach to solve the parameterization problem for huge meshes. Our method is divided into three successive stages:

- ◇ 3D mesh simplification and mesh hierarchy construction (Section 4.1).
- ◇ Parameterization of the simplified, coarse mesh using direct ABF++ (Section 3).
- ◇ Coarse to fine parameterization (Section 4.2).

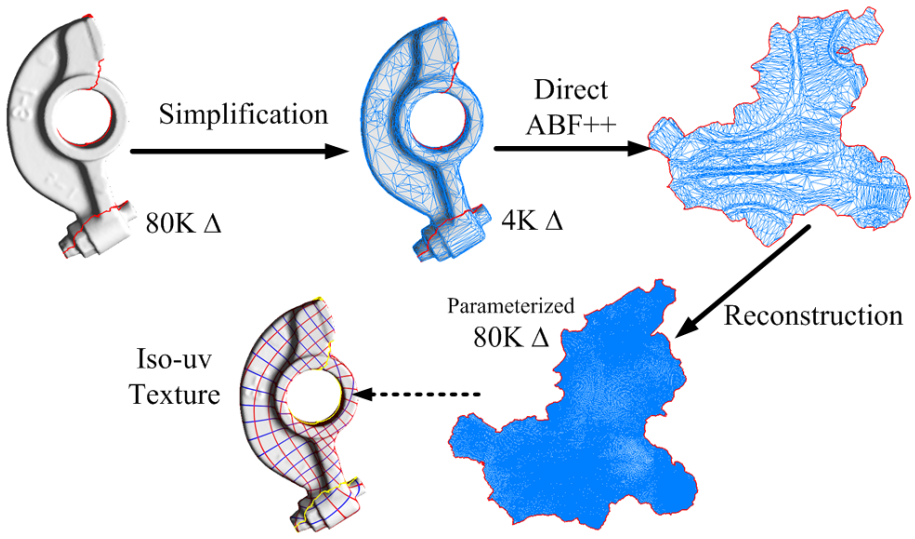


Fig. 3: *Stages of the hierarchical ABF++ algorithm.*

The stages are visualized in Figure 3.

4.1 Mesh Simplification

The mesh simplification is performed through a sequence of edge collapse operations. We preferred the edge collapse simplifier due to the control it provides on the triangle shape in the coarse mesh and the compact simplification history produced during the simplification steps. Similarly to [Ray and Lévy 2003] we use the volume-based geometric error [Lindstrom and Turk 1998] to select the edge to collapse at each iteration. We introduce two modifications to the typical edge selection procedure aimed at reducing the parameterization distortion and speeding up the parameterization process.

The first modification is aimed at avoiding extremely acute and extremely obtuse angles during simplification. Such angles may cause numerical problems during the coarse mesh parameterization as well as during the reconstruction process, slowing down the procedure. Therefore, during simplification we disallow collapses that introduce extreme angles. Based on experiments we set the limit to 2.5°. Thus, after each edge collapse, all the angles in adjoining triangles have to be in the range of $[2.5^\circ, 177.5^\circ]$

The second restriction applies to boundary edge collapses. The reconstruction procedure below introduces less distortion if the boundary of the 2D mesh does not change drastically. To facilitate this we forbid edge collapse operation, collapsing a boundary vertex towards the interior.

During the edge collapse process the simplification history is stored in the form of a list of edge collapse records, sufficient for performing the reverse, vertex split operation.

4.2 Coarse to Fine Parameterization

Following simplification, the resulting coarse mesh is parameterized using the direct ABF++ procedure. The final stage of the algorithm uses this parameterization to compute a param-

eterization of the original mesh using a coarse to fine procedure. This procedure uses the list of edge collapse records stored by the simplification process to add vertices to the parameterized mesh, one at a time. It begins with the simplified 3D mesh and its corresponding coarse parameterization. The original 3D connectivity and the corresponding fine parameterization are constructed by carrying out a sequence of vertex split operations, reversing the simplification process. The splits are performed using the list of edge collapse records, reversing one collapse operation at a time.

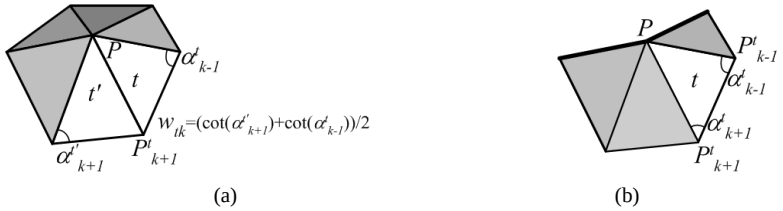


Fig. 4: *Notations used in vertex reconstruction: (a) interior vertex, (b) boundary vertex.*

For each vertex split, the mesh connectivity and the 3D coordinates of the reconstructed vertex are restored based on the edge collapse record. The algorithm then computes a valid parameterization for the current mesh, by computing the 2D position of the reconstructed vertex and adjusting the positions of the adjacent vertices, in this way:

(1) It first sets the 2D coordinates P of the newly reconstructed vertex v using a variation of the DCP method [Desbrun et al. 2002]. The coordinates of an interior vertex are set to

$$P = \frac{1}{\sum w_{tk}} \sum_{(t,k)\in v^*} w_{tk} P^t_{k\oplus 1},$$

where the sum runs over all the triangles adjacent to v, $P^t_{k\oplus 1}$ are the 2D coordinates of the next vertex in the triangle t, and w_{tk} are standard harmonic weights [Eck et al. 1995] (Figure 4). Since the reconstruction is performed one vertex at a time, the 2D coordinates of the adjacent vertices are well defined. The weights w_{tk} are computed on the current 3D mesh. If v is a boundary vertex, the following formula [Desbrun et al. 2002] is used instead

$$\begin{bmatrix} P^x \\ P^y \end{bmatrix} = \frac{\sum_{(t,k)\in v^*} \begin{pmatrix} \cot(\alpha^t_{k\oplus 1}) & 1 & \cot(\alpha^t_{k\ominus 1}) & -1 \\ -1 & \cot(\alpha^t_{k\oplus 1}) & 1 & \cot(\alpha^t_{k\ominus 1}) \end{pmatrix}}{\sum_{(t,k)\in v^*} \cot(\alpha^t_{k\oplus 1}) + \cot(\alpha^t_{k\ominus 1})} \begin{pmatrix} P^t_{k\oplus 1}{}^x \\ P^t_{k\oplus 1}{}^y \\ P^t_{k\ominus 1}{}^x \\ P^t_{k\ominus 1}{}^y \end{pmatrix}, \quad (19)$$

see (Figure 4).

(2) The algorithm checks that the computed vertex placement does not introduce flipped triangles into the surrounding parameterized mesh. If the check fails, the method sets the vertex's 2D coordinates to the center of the kernel of the planar polygon formed by the adjacent vertices. Kernel coordinates do not preserve any parameterization quality, but provide a valid solution with no flipped triangles. For the models we tested, about 0.02% of the vertices (20 out of 100K) were introduced using kernel insertion. Note that we use vertex split for reconstruction. Hence, if the mesh before the split is valid, the triangles adjacent to the vertex being split form a star polygon with non-empty kernel. The split vertex is located in the kernel of this polygon. Hence the

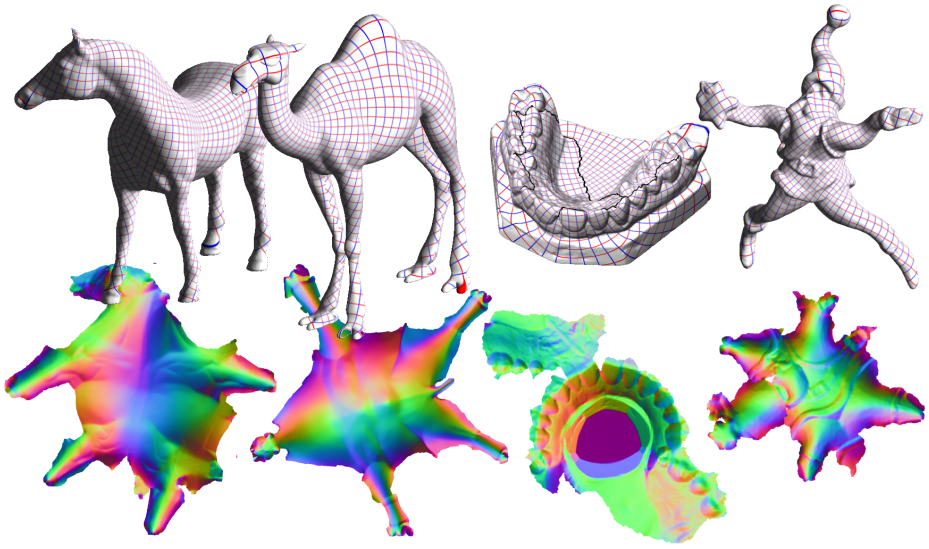


Fig. 5: *Textured and parameterized models. The 2D parameterizations are colored using the 3D normal map.*

sub-polygon into which the reconstructed vertex is introduced, is also a star polygon. Therefore it has a non-empty kernel. Since direct ABF++ generates valid parameterizations, by induction we obtain that a kernel allways exists. And therefore the resulting parameterizations are allways valid. The subsequent local smoothing reduces the parameterization distortion caused by the kernel placement.

(3) Finally, the method performs one iteration of local smoothing. It updates the planar coordinates of the adjacent vertices and then recalculates the coordinates of the current vertex. The new coordinates are computed using the same formulas as in step 1 above. The relaxation is constrained; namely if moving the vertex will cause a triangle flip, the vertex is left in place.

The choice of a simple linear technique for vertex placement and the small number of vertices relocated at each stage make the reconstruction process extremely fast and memory-efficient. Since the vertices of the fine mesh are introduced using only local relaxation, the optimization is less accurate than direct ABF++. However, despite its simplicity, the hierarchical technique provides excellent results, introducing only a minor increase in the parameterization distortion compared to direct ABF++ (Table II).

5. RESULTS AND DISCUSSION

5.1 Examples and Statistics

We tested both the direct and hierarchical methods on a large set of meshes varying from 1K to several million faces (Figures 3, 5, 8, and 6). Closed models were cut into a single topological disc using [Sheffer and Hart 2002]. The direct ABF++ method does not require *any* parameter tuning by the user. The hierarchical ABF++ requires the user to specify a simplification rate. We used a rate of 95% for all but one model (for the 4.MΔ pelvis the rate was 98%). We did not employ the overlap eliminating post-processing prcedure proposed in [Sheffer and de Sturler 2001], hence some overlaps are visible in the resulting models.

Given the details in [Sheffer and de Sturler 2001] implementing this post-processing in the ABF++ framefwork is straightforward.

The results of our runs are summarized in Tables I-III. Table I compares runtimes (when available) for our two methods and other existing parameterization techniques for models with up to 50K faces. For the two ABF++ methods, ABF, and HLSCM [Ray and Lévy 2003], we use our implementation of the methods [Graphite 2003]. The HLSCM method is a hierarchical extension of the free-boundary linear LSCM [Lévy et al. 2002] technique. It is to the best of our knowledge the fastest free-boundary technique available to date. Runtimes for the stretch minimizing method [Sander et al. 2001] were provided by the authors. For the other techniques the times were taken from the respective papers (note that earlier publications, likely used slightly slower machines). Table II compares timing and parametric distortion for parameterization of medium to large models (50K to 230K faces). The original ABF method runs out of physical memory (on 1G RAM machine) for models of above 100K faces. Table III compares the hierarchical ABF++ method and HLSCM for very large meshes. Direct ABF++ fails on meshes this size, due to memory limitations.

We use several metrics to measure the parametric distortion. Similarly to [Sander et al. 2001], we considered the first fundamental form of the mapping (computed per triangle):

$$S(u,v) : R^2 \rightarrow R^3; \quad G(S) \equiv \begin{bmatrix} \frac{\partial S}{\partial u}^2 & \frac{\partial S}{\partial u}\frac{\partial S}{\partial v} \\ \frac{\partial S}{\partial u}\frac{\partial S}{\partial v} & \frac{\partial S}{\partial v}^2 \end{bmatrix}. \tag{20}$$

To measure L_2 stretch we used the same formula as [Sander et al. 2001]. To measure conformality as derived from S we measure L_2 shear:

$$L_2^{shear} = \sqrt{\sum_{t \in T} \left(\frac{\frac{\partial S}{\partial u}\frac{\partial S}{\partial v}}{\left\| \frac{\partial S}{\partial u} \right\| \left\| \frac{\partial S}{\partial v} \right\|} \right)^2 \frac{area3D(t)}{\sum_{t \in T} area3D(t)}}. \tag{21}$$

The second component of conformality, the equal stretch property $\frac{\partial S}{\partial u}^2 \equiv \frac{\partial S}{\partial v}^2$, was measured as described in [Mogilnitsky 2004]. For all the compared techniques the error was negligible (less than 0.003). Hence we did not include it in the tables. Similarly to [Sheffer and de Sturler 2001] we also measured the angular distortion $\frac{E(\alpha)}{3n_f}$ (Equation 1) for the different parameterizations.

The timing for all but one model are computed on a 1.7GHz Pentium M (1G RAM) machine. For the largest model, the pelvis (4.2M faces) a slightly stronger PIV 2.4GHz machine was used.

The runtime comparison demonstrates that direct ABF++ is up to an order of magnitude faster than the standard version. Moreover, for all the examples with above 50K faces the standard ABF fails to accurately reconstruct the coordinates from the angles due to numerical problems. The shear introduced by both direct ABF++ and HLSCM is very low, with one method performing better on some models, and the second on others. While direct ABF++ is slower than HLSCM, it introduces significantly less stretch, an important advantage for many applications.

For all the examples, hierarchical ABF++ is significantly faster than HLSCM. Compared to the direct ABF++ implementation it slightly increases the distortion (in terms of both stretch and shear). However, it still introduces an order of magnitude less stretch than

	1KΔ	6KΔ	10K Δ	50KΔ
hierarchical ABF++	0.14	0.9	1.5	3.5
direct ABF++	0.3	1.5	3	16
ABF	1.3	10	27	125
HLSCM [Ray and Lévy 2003]	0.03	0.2	0.5	16.5
Zayer et. al [Zayer et al. 2004]	1	30	110	NA
MIPS [Hormann and Greiner 2000]	20	NA	NA	NA
Degener et. al [Degener et al. 2003]	NA	33	50	300
Stretch [Sander et al. 2001]	NA	8.5	19.3	52

Table I: *Timing comparison (in seconds), when available, for direct/hierarchical ABF++ and other free-boundary techniques for small to medium models.*

Model	Number of Faces	Method	Runtime (sec)	Shear	Stretch	Angular Distortion
camel	78,144	HABF++	6	0.0463	1.6019	0.000129
		DABF++	46	0.0235	1.4864	6.43e-5
		ABF	690			
		HLSCM	34	0.0282	13.7945	0.00017
		Stretch	127	0.22683	1.0534	0.0518
rocker arm	80,354	HABF++	5	0.0238	1.0897	3.86e-5
		DABF++	46	0.0153	1.0905	2.3e-5
		ABF	976			
		HLSCM	36	0.0160	3.9198	2.71e-5
horse	96,966	HABF++	7	0.0398	1.3851	4.81e-5
		DABF++	84	0.0401	1.3981	3.22e-5
		ABF	FAIL			
		HLSCM	44	0.0299	31.6283	6.32e-5
santa	151,558	HABF++	10	0.0178	1.1623	2.2e-5
		DABF++	71	0.0137	1.1378	1.59e-5
		ABF	FAIL			
		HLSCM	71	0.0166	140.7869	2.02e-5
teeth	233,204	HABF++	18	0.0227	1.4166	2.08e-5
		DABF++	538	0.020	1.4771	1.62e-5
		ABF	FAIL			
		HLSCM	111	0.0466	165.	0.0339

Table II: *Parameterization comparison of standard ABF, hierarchical LSCM (HLSCM), direct (DABF++) and hierarchical (HABF++) ABF++ for medium to large models. For the camel model we also added the statistics for the method of Sander et. al, to highlight the difference between stretch preserving and conformal techniques. The shear (conformality) error is roughly identical for all four conformal methods. The angle based methods, however, generate significantly less stretch. Note that the optimal value for stretch is 1. While direct ABF++ is slower than HLSCM, its hierarchical version is actually significantly faster. We do not provide error metrics for ABF since, for all the above examples the reconstruction procedure used by ABF failed (due to numerical problems). ABF ran out of memory for the horse and larger models.*

HLSCM, as demonstrated by the texture in Figure 6. The distortion increases with larger simplification rates (Figure 7). For inputs of 200K+ faces a 95% simplification rate means that the coarse level mesh contains 10K+ triangles. Parameterizing the coarse mesh using the standard ABF would take about 30sec (Table I). In contrast, using direct ABF++ the

Model	Number of Faces	Algorithm	Runtime (sec)	Shear	Stretch	Angular Distortion
bust	605,846	HABF++	107	0.0298	241.05	2.51e-5
		HLSCM	305	0.042	1280	0.0042
David	698,572	HABF++	50	0.0386	1.8954	3.12e-5
		HLSCM	351	0.065	42.4	0.0060
pelvis	4,241,328	HABF++	819	0.0077	1.2001	3.4e-6
		HLSCM	FAIL			

Table III: *Parameterization statistics for large models: hierarchical ABF++ and HLSCM. The direct ABF++ runs out of memory for this size of models.*

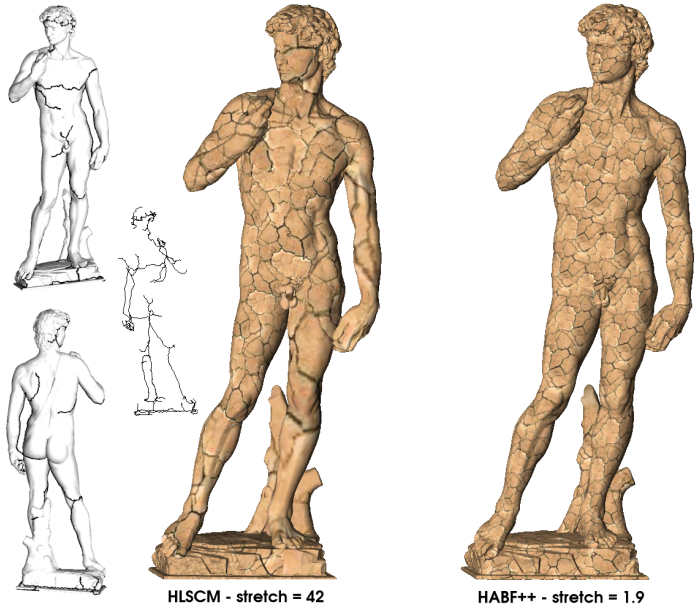
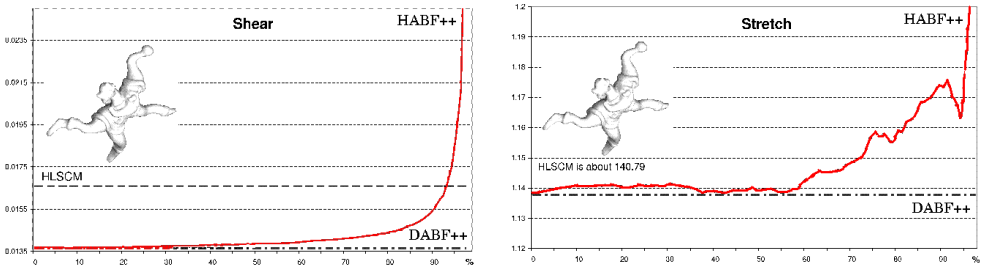


Fig. 6: *Texture mapping the David model (700K Δ). (left) The seams used for cutting the model; (center) texturing using HLSCM; (left) texturing using HABF++. Notice the extreme difference in stretch.*

parameterization of the coarse 10K mesh takes 3 seconds and the hierarchical parameterization of the full 200K face mesh (including reconstruction) takes only 18sec (Table II). Therefore, even within the hierarchical framework, it is impractical to use the standard ABF instead of direct ABF++.

Both the direct and hierarchical ABF++ methods remain stable even on meshes with very high stretch (which cause huge scale variations in the parameterized mesh), as demonstrated by Figure 8.

The last example, Figure 9, demonstrates the application of our method to generation of a normal-mapped simplified model of the pelvis. The original model has 4.2M triangles. Our hierarchical method generates a conformal single-chart parameterization of this model in about 14min (Figure 9 (a)). The simplified normal-mapped model has only 8K triangles, but using the normal map the visualization preserves all the details of the original model (Figure 9 (c)). To generate a more compact texture space, the parameterized (2D) mesh was cut manually and packed into a square domain using [Lévy et al. 2002].



(a) (b)

Fig. 7: *Impact of simplification rate of hierarchical ABF++ on parametric distortion: (a) shear, (b) stretch. For the 95% rate that we use, we get an increase of 0.004 in shear and 0.025 in stretch compared to the direct method.*

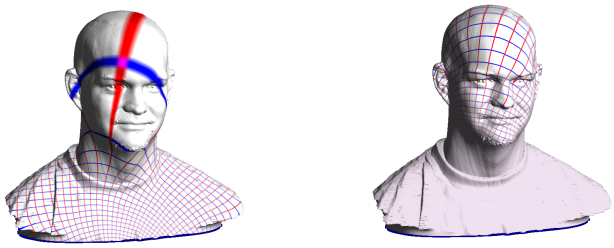


Fig. 8: *Hierarchical ABF++ for huge model (605K) with very high area to perimeter ratio. For angle preserving mappings such as ABF++ this results in extreme stretch (241.1). The stability of the method is not affected. The two images show the iso-lines on the surface at different frequency, to highlight the orthogonality preservation at all the points on the surface. By comparison the stretch introduced by HLSCM was about five times larger (1280).*

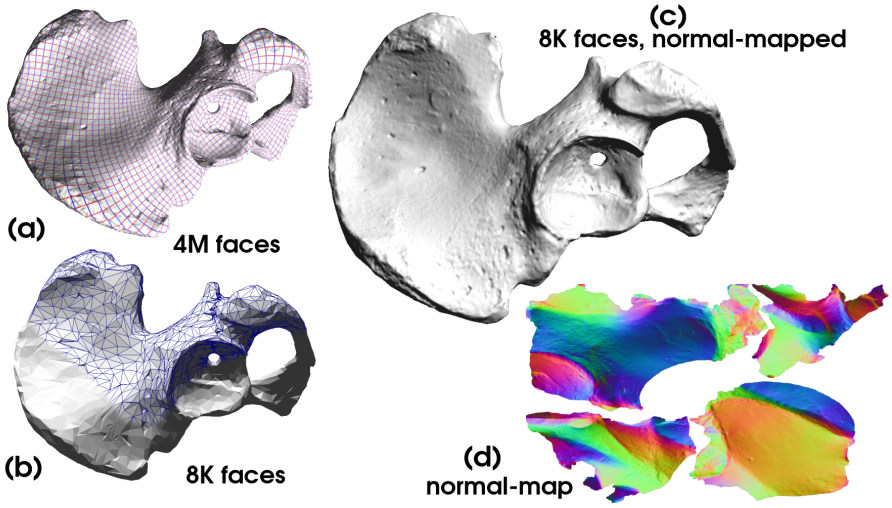


Fig. 9: *Our method applied to generate normal-mapped simplified model of a pelvis (4.2MΔ).*

5.2 Discussion

The numerical and visual comparisons throughout the paper consistently indicate that ABF and its extensions create angle preserving parameterizations with significantly lower stretch than linear angle preserving methods (LSCM/DCP). In this section we conjecture as to the reasons for this difference. In this discussion we will consider the LSCM formulation [Lévy et al. 2002], however we expect the same argument to apply to DCP [Desbrun et al. 2002], due to the equivalence between the methods [Lévy et al. 2003]. The main observation to remenber is that in contrast to the continuous case, for most 3D meshes there exists no truly conformal (angle preserving) mapping to 2D (the sum of angles around a vertex in 2D must be 2π, while the sum can be arbitrary in 3D). Hence, the minimum of a functional measuring the conformality of a mapping will not be zero for most meshes. As a result the choice of different functional formulations can lead to very different minimizers.

LSCM is based on the observation that if a function is conformal, so is its inverse. Hence, LSCM minimizes a metric of the conformal energy of the inverse (3D to 2D) parameterization. The metric is quadratic and therefore easy to minimize. The energy functional defined by LSCM per triangle is proportional to the area of the triangle in (u,v) space. Therefore, the energy can be reduced by reducing area. LSCM avoids degenerate configurations by pinning two vertices, making the energy functional postive-definite. The solution depends on the vertices pinned. It appears that to reduce the energy functional, LSCM reduces the 2D area of triangles in regions of high distortion. This causes the extreme stretch we observed.

This concurs with the observation made by Hormann and Greiner [Hormann and Greiner 2000] explaining why using Dirichlet energy to minimize/measure parametric distortion is sub-optimal. Instead they measure Dirichlet energy per parameter-space area. This scaling prevents triangle shrinkage (but makes the energy non-linear and hence difficult to minimize).

The ABF formulation is based on pure angular quantities, and is independent of tringle size in the 3D or 2D space. Hence, similarly to MIPS it does not suffer from triangle shrinkage and hence introduces significantly less stretch than LSCM/DCP.

6. SUMMARY

We have presented a new robust and scalable conformal parameterization technique. The first component of this technique, the direct ABF++ method, is applicable to meshes of up to 300K faces. It is faster than existing non-linear free-boundary techniques, parameterizing meshes of 100K+ faces in 1 to 2 minutes. The hierarchical version of the method, scales to meshes of millions of triangles. Both methods compute provably valid, conformal parameterizations with very low stretch. The hierarchical ABF++ is faster than any existing free-boundary technique we are familiar with.

Acknowledgements

We would like to thank Stanford University and cyberware.com for providing the data-sets used in this paper. Special thanks to the Graphite team for providing the software. This research was performed with the support of the AIM@SHAPE EU Network of excellence and NSERC.

REFERENCES

AKSOYLU, B., KHODAKOVSKY, A., AND SCHRÖDER, P. Multilevel solvers for unstructured surface meshes. *SIAM J. Sci. Comput, in review*.

ALLIEZ, P., STEINER, D. C., DEVILLERS, O., LEVY, B., AND DESBRUN, M. 2003. Anisotropic Mesh Remeshing. *ACM TOG (SIGGRAPH)*.

DEGENER, P., MESETH, J., AND KLEIN, R. 2003. An adaptable surface parametrization method. In *12th International Meshing Roundtable 2003*.

DEMMEL, J. W., EISENSTAT, S. C., GILBERT, J. R., LI, X. S., AND LIU, J. W. H. 1999. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications 20*, 3, 720–755.

DESBRUN, M., MEYER, M., AND ALLIEZ, P. 2002. Intrinsic parameterizations of surface meshes. In *Proceedings of Eurographics*. 209–218.

ECK, M., DEROSE, T., DUCHAMP, T., HOPPE, H., LOUNSBERY, M., AND STUETZLE, W. 1995. Multiresolution Analysis of Arbitrary Meshes. In *Computer Graphics (SIGGRAPH Conf. Proc.)*. ACM, 173–182.

FLOATER, M. 1997. Parametrization and smooth approximation of surface triangulations. *CAGD 14*, 3 (April), 231–250.

FLOATER, M. S. 2003. Mean value coordinates. *CAGD 20*, 19–27.

FLOATER, M. S. AND HORMANN, K. 2004. Surface parameterization: a tutorial and survey. In *Advances on Multiresolution in Geometric Modelling*, M. S. F. N. Dodgson and M. Sabin, Eds. Springer-Verlag, Heidelberg.

GRAPHITE. 2003. http://www.loria.fr/ levy/Graphite/index.html.

GU, X., GORTLER, S., AND HOPPE, H. 2002. Geometry images. *ACM TOG (SIGGRAPH)*, 355–361.

GU, X. AND YAU, S.-T. 2002. Computing conformal structures of surfaces. *Communications in information and systems 2*, 121–146.

HORMANN, K. 2001. Theory and applications of parameterizing triangulations. Ph.D. thesis, Department of Computer Science, University of Erlangen.

HORMANN, K. AND GREINER, G. 2000. MIPS: An efficient global parametrization method. In *Curve and Surface Design: Saint-Malo 1999*, P.-J. Laurent, P. Sablonnière, and L. Schumaker, Eds. Vanderbilt University Press, 153–162.

LEE, A. W. F., SWELDENS, W., SCHRODER, P., COWSAR, L., AND DOBKIN, D. 1998. MAPS: Multiresolution adaptive parameterization of surfaces. In *Computer Graphics (SIGGRAPH)*. ACM, 95–104.

LEVY, B. 2003. Dual Domain Extrapolation. *ACM TOG (SIGGRAPH)*.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2002. Least Squares Conformal Maps. *ACM TOG (SIGGRAPH)*.

LÉVY, B., PETITJEAN, S., RAY, N., AND MAILLOT, J. 2003. http://www.loria.fr/ levy/erratum.html.

LIESEN, J., DE STURLER, E., SHEFFER, A., AYDIN, Y., AND SIEFERT, C. 2001. Preconditioners for indefinite linear systems arising in surface parameterization.

LINDSTROM, P. AND TURK, G. 1998. Fast and memory efficient polygonal simplification. In *Proceedings of IEEE Visualization*. 279–286.

MAILLOT, J., YAHIA, H., AND VERROUST, A. 1993. Interactive texture mapping. In *SIGGRAPH 93 Conf. Proc.* Addison Wesley, 27–34.

MOGILNITSKY, M. 2004. Efficient, low distortion, conformal parameterization of large meshes. *MSc Thesis, Technion*.

NOCEDAL AND WRIGHT. 2000. *Numerical Optimization*. Springer.

PINKALL, U. AND POLTHIER, K. 1993. Computing discrete minimal surfaces and their conjugates. *Experimental Math. 2*, 15.

PRAUN, E., SWELDENS, W., AND SCHRODER, P. 2001. Consistent mesh parameterization. In *Computer Graphics (SIGGRAPH)*. ACM, 179–184.

RAY, N. AND LÉVY, B. 2003. Hierarchical least squares conformal maps. In *11th Pacific Conference on Computer Graphics and Applications 2003 - PG'03, Canmore, Canada*. 263–270.

SANDER, P., GORTLER, S., SNYDER, J., AND HOPPE, H. 2002. Signal-specialized parametrization. In *Eurographics Workshop on Rendering*. 87–100.

SANDER, P., SNYDER, J., GORTLER, S., AND HOPPE, H. 2001. Texture mapping progressive meshes. In *Computer Graphics (SIGGRAPH)*. ACM, 409–416.

SHEFFER, A. AND DE STURLER, E. 2001. Parameterization of faceted surfaces for meshing using angle based flattening. *Engineering with Computers 17*, 326–337.

SHEFFER, A. AND HART, J. 2002. Seamster: Inconspicuous low-distortion texture seam layout. In *IEEE Visualization*.

SORKINE, O., COHEN-OR, D., AND TOLEDO, S. 2003. High-pass quantization for mesh encoding. In *Proceedings of the Eurographics/ACM SIGGRAPH symposium on Geometry processing*. Eurographics Association, 42–51.

SUMNER, R. W. AND POPOVIC, J. 2004. Deformation transfer for triangle meshes. *ACM TOG (Proc.SIGGRAPH 04) 23*, 3, 397–403.

TUTTE, W. 1960. Convex representation of graphs. In *Proc. London Math. Soc.* Vol. 10.

YOSHIZAWA, S., BELYAEV, A., AND SEIDEL, H.-P. 2004. A fast and simple stretch-minimizing mesh parameterization. In *Proc. Shape Modeling and Applications*. 200–208.

ZAYER, R., ROESSL, C., AND SEIDEL, H. P. 2004. Variations on angle based flattening. *Proceedings Mingle Workshop*.